require：加载模块

module（模块）:在 Node.js 模块系统中，每个文件都被视为一个独立的模块。

访问主模块:当 Node.js 直接运行一个文件时， require.main 会被设为它的 module。 这意味着可以通过 require.main === module 来判断一个文件是否被直接运行

包管理器得技巧: Node.js 的 require() 函数的语义被设计得足够通用化，可以支持许多合理的目录结构。即便存在循环依赖或依赖冲突，每个模块还是可以获得它所依赖的包的一个可用版本。

.mjs 扩展名:不可以 require() 具有 .mjs 扩展名的文件。 试图这样做会抛出[错误](http://nodejs.cn/s/RrL8jT" \t "_blank)。 .mjs 扩展名是保留给 [ECMAScript 模块](http://nodejs.cn/s/iyYkEg" \t "_blank)，无法通过 require() 加载。

缓存:模块在第一次加载后会被缓存。 这也意味着（类似其他缓存机制）如果每次调用 require('foo') 都解析到同一文件，则返回相同的对象。多次调用 require(foo) 不会导致模块的代码被执行多次。 这是一个重要的特性。 借助它, 可以返回“部分完成”的对象，从而允许加载依赖的依赖, 即使它们会导致循环依赖。如果想要多次执行一个模块，可以导出一个函数，然后调用该函数

模块缓存得注意事项:模块是基于其解析的文件名进行缓存的。 由于调用模块的位置的不同，

模块可能被解析成不同的文件名（比如从 node\_modules 目录加载），这样就不能保证 require('foo') 总能返回完全相同的对象。

此外，在不区分大小写的文件系统或操作系统中，被解析成不同的文件名可以指向同一文件，但缓存仍然会将它们视为不同的模块，并多次重新加载。

核心模块:Node.js 有些模块会被编译成二进制。核心模块定义在 Node.js 源代码的 lib/ 目录下。require() 总是会优先加载核心模块。 例如， require('http') 始终返回内置的 HTTP 模块，即使有同名文件。

循环:当循环调用 require() 时，一个模块可能在未完成执行时被返回。

当 main.js 加载 a.js 时， a.js 又加载 b.js。 此时， b.js 会尝试去加载 a.js。 为了防止无限的循环，会返回一个 a.js 的 exports 对象的 未完成的副本 给 b.js 模块。

然后 b.js 完成加载，并将 exports 对象提供给 a.js 模块。

当 main.js 加载这两个模块时，它们都已经完成加载。

文件模块：如果按确切的文件名没有找到模块，则 Node.js 会尝试带上 .js、 .json 或 .node 拓展名再加载。目录作为：可以把程序和库放到一个单独的目录，然后提供一个单一的入口来指向它。 把目录递给 require() 作为一个参数，有三种方式。

1. 第一种方式是在根目录下创建一个 package.json 文件
2. 如果目录里没有 package.json 文件，或者 'main' 入口不存在或无法解析，则 Node.js 将会试图加载目录下的 index.js 或 index.node 文件。
3. 如果这些尝试失败，则 Node.js 将会使用默认错误报告整个模块的缺失

从 node\_modules 目录加载：如果传递给 require() 的模块标识符不是一个[核心模块](http://nodejs.cn/api/modules.html" \l "modules_core_modules)，也没有以 '/' 、 '../' 或 './' 开头，则 Node.js 会从当前模块的父目录开始，尝试从它的 /node\_modules 目录里加载模块。 Node.js 不会附加 node\_modules 到一个已经以 node\_modules 结尾的路径上。如果还是没有找到，则移动到再上一层父目录，直到文件系统的根目录。

从全局目录加载：如果 NODE\_PATH 环境变量被设为一个以冒号分割的绝对路径列表，则当在其他地方找不到模块时 Node.js 会搜索这些路径。在 Windows 系统中， NODE\_PATH 是以分号（;）间隔的。在当前的[模块解析](http://nodejs.cn/s/mXg3yX" \t "_blank)算法被定义之前， NODE\_PATH 最初是创建来支持从不同路径加载模块的。强烈建议将所有的依赖放在本地的 node\_modules 目录。 这样将会更快地加载，且更可靠。

模块封装器: 在执行模块代码之前，Node.js 会使用一个如下的函数封装器将其封装：

(function(exports,require,module,\_filename,\_dirname){

//模块的代码实际上在这里

});

通过这样做，Node.js 实现了以下几点：

它保持了顶层的变量（用 var、 const 或 let 定义）作用在模块范围内，而不是全局对象。

它有助于提供一些看似全局的但实际上是模块特定的变量，例如：实现者可以用于从模块中导出值的 module 和 exports 对象。

包含模块绝对文件名和目录路径的快捷变量 \_\_filename 和 \_\_dirname 。

模块作用域:\_\_dirname当前模块的目录名。 与 [\_\_filename](http://nodejs.cn/s/RH6qCV" \t "http://nodejs.cn/api/_blank) 的 [path.dirname()](http://nodejs.cn/s/7t4KCq" \t "http://nodejs.cn/api/_blank) 相同。

\_\_filename当前模块的文件名。 这是当前的模块文件的绝对路径（符号链接会被解析）。

Exports

exports 变量是在模块的文件级作用域内可用的，且在模块执行之前赋值给 module.exports。

Module对象

在每个模块中， module 的自由变量是对表示当前模块的对象的引用。 为方便起见，还可以通过全局模块的 exports 访问 module.exports。 module 实际上不是全局的，而是每个模块本地的。

 module.exports 用于指定一个模块所导出的内容，即可以通过 require() 访问的内容。

require(id)

id [<string>](http://nodejs.cn/s/9Tw2bK" \t "http://nodejs.cn/api/_blank) 模块的名称或路径。

返回: [<any>](http://nodejs.cn/s/6sTGdS" \t "http://nodejs.cn/api/_blank) 导入的模块内容。

用于引入模块、 JSON、或本地文件。 可以从 node\_modules 引入模块。

require.cache

被引入的模块将被缓存在这个对象中。

require.resolve(request[, options])

request [<string>](http://nodejs.cn/s/9Tw2bK" \t "http://nodejs.cn/api/_blank) 需要解析的模块路径。

options [<Object>](http://nodejs.cn/s/jzn6Ao" \t "http://nodejs.cn/api/_blank)

返回: [<string>](http://nodejs.cn/s/9Tw2bK" \t "http://nodejs.cn/api/_blank)

require.resolve.paths(request)

request [<string>](http://nodejs.cn/s/9Tw2bK" \t "http://nodejs.cn/api/_blank) 被查询解析路径的模块的路径。

返回: [<string[]>](http://nodejs.cn/s/9Tw2bK" \t "http://nodejs.cn/api/_blank) | [<null>](http://nodejs.cn/s/334hvC" \t "http://nodejs.cn/api/_blank)

返回一个数组，其中包含解析 request 过程中被查询的路径，如果 request 字符串指向核心模块（例如 http 或 fs）则返回 null。

module.children

被该模块引用的模块对象。

module.exports

module.exports 对象由 Module 系统创建。

exports 快捷方式

module.exports.f = ... 可以更简洁地写成 exports.f = ...

module.filename

模块的完全解析后的文件名。

module.id

模块的标识符。 通常是完全解析后的文件名。

module.loaded

[<boolean>](http://nodejs.cn/s/jFbvuT" \t "http://nodejs.cn/api/_blank)

模块是否已经加载完成，或正在加载中。

module.parent

[<module>](http://nodejs.cn/s/pqs7NL" \t "http://nodejs.cn/api/_blank)

最先引用该模块的模块。

module.paths

[<string[]>](http://nodejs.cn/s/9Tw2bK" \t "http://nodejs.cn/api/_blank)

模块的搜索路径。

module.require(id)[#](http://nodejs.cn/api/modules.html" \l "modules_module_require_id)

id [<string>](http://nodejs.cn/s/9Tw2bK" \t "http://nodejs.cn/api/_blank)

返回: [<any>](http://nodejs.cn/s/6sTGdS" \t "http://nodejs.cn/api/_blank) 导出的模块内容。

module.require() 方法提供了一种加载模块的方法，就像从原始模块调用 require() 一样。

为了做到这个，需要获得一个 module 对象的引用。 因为 require() 会返回 module.exports，且 module 通常只在一个特定的模块代码中有效，所以为了使用它，必须显式地导出。

Module 对象[#](http://nodejs.cn/api/modules.html" \l "modules_the_module_object_1)

[<Object>](http://nodejs.cn/s/jzn6Ao" \t "http://nodejs.cn/api/_blank)

为 Module 实例提供通用方法。 module 变量常见于文件模块中。 通过 require('module') 获取。

module.builtinModules[#](http://nodejs.cn/api/modules.html" \l "modules_module_builtinmodules)

[<string[]>](http://nodejs.cn/s/9Tw2bK" \t "http://nodejs.cn/api/_blank)

罗列 Node.js 提供的所有模块名称。可以用来判断模块是否为第三方所维护。

module.createRequire(filename)[#](http://nodejs.cn/api/modules.html" \l "modules_module_createrequire_filename)

filename [<string>](http://nodejs.cn/s/9Tw2bK" \t "http://nodejs.cn/api/_blank) | [<URL>](http://nodejs.cn/s/5dwq7G" \t "http://nodejs.cn/api/_blank) 用于构造 require 函数的文件名。必须是一个文件 URL 对象、文件 URL 字符串、或绝对路径字符串。

返回: [<require>](http://nodejs.cn/s/bVPMwV" \t "http://nodejs.cn/api/_blank) require 函数。

module.syncBuiltinESMExports（）[＃](http://nodejs.cn/api/modules.html" \l "modules_module_syncbuiltinesmexports)

该module.syncBuiltinESMExports()方法更新内置ES模块的所有实时绑定，以匹配CommonJS导出的属性。它不会在ES模块中添加或删除导出的名称。